CLAIMS:

1. A data structure depicting one or more queues storing data to be routed by a unicast scheduler, said data structure comprising:

a Structure Pointer memory comprising multiple addressable records, each record storing a pointer to a location in memory of a segment which is a part of a packet associated with a respective queue, a Structure Pointer pointing to a record in the Structure Pointer memory associated with a successive segment of the packet in the queue, a packet indicator indicating whether the segment is a first segment and/or a last segment in the packet,

a Head & Tail memory comprising multiple addressable records, each record storing for a respective queue a corresponding address in the Structure Pointer memory of the first and last packets in the queue, and

a free structure memory comprising multiple addressable records, each record pointing to a next available memory location in the Structure Pointer memory.

2. The data structure according to Claim 1, adapted to depict one or more queues storing data to be routed by a multicast schedulerand further comprising:

a multiplicity memory comprising multiple addressable records, each record storing a value corresponding to a number of destinations to which a respective queue is to be routed.

3. An enqueue processor adapted to append a new data packet to a queue having the data structure according to Claim 1.

4. An enqueue processor adapted to append a new data packet to a queue having the data structure according to Claim 2.

5. A grant processing unit adapted to receive requests for data departing from a granted queue having the data structure according to Claim 1, said grant processing unit comprising one or more dequeue processors, each adapted to handle a single grant at a time for removing data from said granted queue.

6. The grant processing unit according to Claim 5, wherein each dequeue processor maintains a respective database that includes for each queue a registered snapshot head and tail pointer, an *In_process* flag that when set indicates that the respective queue is in a dequeue process, and a *Touched* flag that when set indicates that one or more structures have been added to the respective queue.

7. A grant processing unit adapted to receive requests for data departing from a granted queue having the data structure according to Claim 2, said grant processing unit comprising one or more dequeue processors, each adapted to handle a single grant at a time for removing data from said granted queue.

8. The grant processing unit according to Claim 7, wherein each dequeue processor maintains a respective database that includes for each queue a registered snapshot head and tail pointer, an *In_process* flag that when set indicates that the respective queue is in a dequeue process, and a *Touched* flag that when set indicates that one or more structures have been added to the respective queue.

9. A method for receiving and dispatching data packet segments associated with one or more unicast queues, the method comprising:

(a) storing received packets, segment by segment, each associated with said queues in a data structure that is adapted to manage data packets as linked lists of segments, in the following manner:

i) for each arriving segment, fetching a structure pointer from a free structure reservoir, and fetching a data segment pointer from a free data pointer reservoir;

ii) storing the data segment in a memory address pointed to by said data segment pointer;

iii) storing the data segment pointer in the structure pointed to by the structure pointer;

iv) maintaining a packet indicator in the data structure for indicating if the current segment is a first segment or a last segment or an intermediate segment in the packet;

v) appending the data structure to a structure linked list associated with said queue;

(b) dispatching stored packet train comprising a specified number of segments, segment by segment, from a specified queue using the following steps:

i) creating a snapshot of the linked list associated with said specified queue by copying the list head and tail structure pointers to a snapshot head and snapshot tail pointers;

ii) fetching a data segment pointer from the structure pointed to by the snapshot head pointer,

iii) dispatching a current data segment pointed to by said data segment pointer;

iv) updating the snapshot head pointer to point to a successive structure in the linked list;

v) repeating **(ii)** to **(iv)** until the packet indicator of the current segment indicates that the current segment is the end of packet, and dispatching all segments of a successive packet in the queue would result in dispatching more data segments than said specified number of segments;

vi) concurrent with stages **ii)** to **v)**, allowing reception of segments of newly arrived packets to continue, according to the following measures:

(1) upon arrival of a first segment, initializing the linked list of the specified queue;

(2) storing and managing segments according to stages **(a) i)** to **v)**;

vii) upon completion of stage **(b) v)**, concatenating segments as follows:

(1) if no new segments have arrived, copying the snapshot head and tail pointers to the queue linked list head and tail pointers;

(2) if the snapshot linked list were completely emptied, preserving the queue linked list, and holding only the newly arriving segments;

(3) otherwise, concatenating the linked list of the newly arrived segments to the snapshot linked list.

10. A method for receiving and dispatching data packets associated with one or more unicast queues, the method comprising:

(a) storing data associated with said queues in a data structure that is configured to include:

i) a Structure Pointer memory comprising multiple addressable records, each record storing a pointer to a location in memory of a segment of a packet associated with a respective queue, a Structure Pointer pointing to a record in the Structure Pointer memory associated with a successive segment in the queue, a packet indicator indicating whether the segment is a first segment and/or a last segment in the packet,

ii) a Head & Tail memory comprising multiple addressable records, each record storing for a respective queue a corresponding address in the Structure Pointer memory of the first and last packets in the queue, and

iii) a free structure memory comprising multiple addressable records, each record pointing to a next available memory location in the Structure Pointer memory;

(b) maintaining for each queue a respective database that includes for each queue a registered snapshot head and tail pointer, an *In_process* flag, and a *Touched* flag;

(c) on one or more segments of incoming data packets arriving at a queue:

i) reading the free structure memory to determine a next available record in the Structure Pointer memory for storing therein data relating to the incoming packet;

ii) storing data pertaining to the incoming packet in the Structure Pointer memory at the next available record, as follows:

- if an incoming segment is a first segment in the packet:

1) setting the packet indicator to indicate that the incoming segment is the first and last segment in the packet;

- if the incoming segment is not the first segment and not the last segment in the packet:

2) setting the packet indicator to indicate that the incoming segment is an intermediate segment in the packet,

- if the incoming segment is the last segment in the packet:

3) setting the packet indicator to indicate that the incoming segment is the last segment in the packet,

4) setting the Structure Pointer to NULL, and

5) if the current record is not the first record, then setting the Structure Pointer of a preceding record to point to the current record;

iii) updating a respective record of the Head & Tail memory corresponding to said queue;

iv) updating the free structure memory to point to an available record; and

v) setting the *Touched* flag;

(d) upon reception of a grant identifying a granted queue from which a specified number of outgoing data packets should depart:

i) setting the *In_process* flag when the respective queue is in a dequeue process;

ii) reading a respective record of the Head & Tail memory corresponding to the granted queue and registering corresponding data in the snapshot Head record and the snapshot Tail record;

iii) recovering data at a corresponding record in the Structure Pointer memory pertaining to the snapshot Head record, updating the head using the next structure record of the recovered data, and sending the data pointer of the recovered data to an external module for fetching the data segment pointed to by the data pointer,

iv) updating a respective record of the Head & Tail memory corresponding to said queue; and

v) updating the free structure memory so to add a pointer to the record in the Structure Pointer memory vacated by the outgoing data packet;

vi) repeating stages **iii)** to **vii)** until one the following occurs:

1) the snapshot Head becomes equal to the snapshot Tail; or

2) the number of departing packets reaches a prescribed number of packet as given in the grant.

11. The method according to Claim 10, wherein the data structure is adapted to depict one or more queues storing data to be routed by a multicast scheduler and further comprises:

a multiplicity memory comprising multiple addressable records, each record storing a value corresponding to a number of destinations to which a respective queue is to be routed;

said method further comprising:

incrementing a respective record of the multiplicity memory corresponding to said queue on one or more incoming data packets arriving at a queue; and

upon reception of a grant if a respective record of the multiplicity memory corresponding to said queue is greater than unity, decrementing said record.

12. The method according to Claim 10, including:

i) maintaining an index of the granted queue;

ii) initializing the snapshot head and tail pointers to hold a snapshot of the granted queue;

iii) setting the *In_process* flag at the beginning of a grant to indicate that the granted queue is now accessed by a dequeue process;

iv) at the beginning of a dequeue process, upon reception of one or more data segments:

1) resetting the *Touched* flag to indicate that a new structure containing one or more data segments has been added to the queue;

2) writing head of the queue in the head & tail RAM with the pointer associated with the structure that is received first; and

3) diverting the tail of the queue to point to the structure that is received last;

v) during subsequent stages where new structures enter the queue and the *Touched* flag is set, updating the tail of the queue with the new structures thereby setting the queue's *Touched* flag; and

vi) upon termination of the grant re-setting the *In_process* flag.

13. The method according to Claim 11, further including concatenating the snapshot linked list with the queue linked list upon termination of a dequeue process.

14. The method according to Claim 13, wherein said concatenating comprises:

i) setting two temporary values, temp_Head and temp_Tail, as follows;

1) initializing the temporary values temp_Head and temp_Tail to the values of the granted queue head and tail values taken from the head & tail RAM, respectively;

2) if the queue were cleared, setting both temp_Head and temp_Tail to NULL;

3) if the queue were not cleared, maintaining the value of temp_Tail, and setting temp_Head to point to the last structure, which was not released;

ii) concatenating as follows:

4) if the queue were not touched by the enqueue process, setting both head and tail values of the head & tail RAM to the values of temp_Head and temp_Tail, respectively;

5) if the queue were touched by the enqueue process, and the snapshot queue has not yet been cleared, updating the queue head to the value of temp_Head, and maintaining value of the queue tail;

6) if the queue were touched by the enqueue process, and the snapshot queue was cleared, maintaining the values of both head and tail.

15. A program storage device readable by machine, tangibly embodying a program of instructions executable by the machine to perform method steps for receiving and dispatching data packet segments associated with one or more unicast queues, the method comprising:

(a) storing received packets, segment by segment, each associated with said queues in a data structure that is adapted to manage data packets as linked lists of segments, in the following manner:

i) for each arriving segment, fetching a structure pointer from a free structure reservoir, and fetching a data segment pointer from a free data pointer reservoir;

ii) storing the data segment in a memory address pointed to by said data segment pointer;

iii) storing the data segment pointer in the structure pointed to by the structure pointer;

iv) maintaining a packet indicator in the data structure for indicating if the current segment is a first segment or a last segment or an intermediate segment in the packet;

v) appending the data structure to a structure linked list associated with said queue;

(b) dispatching stored packet train comprising a specified number of segments, segment by segment, from a specified queue using the following steps:

i) creating a snapshot of the linked list associated with said specified queue by copying the list head and tail structure pointers to a snapshot head and snapshot tail pointers;

ii) fetching a data segment pointer from the structure pointed to by the snapshot head pointer,

iii) dispatching a current data segment pointed to by said data segment pointer;

iv) updating the snapshot head pointer to point to a successive structure in the linked list;

v) repeating **(ii)** to **(iv)** until the packet indicator of the current segment indicates that the current segment is the end of packet, and dispatching all segments of a successive packet in the queue would result in dispatching more data segments than said specified number of segments;

vi) concurrent with stages **ii)** to **v)**, allowing reception of segments of newly arrived packets to continue, according to the following measures:

(1) upon arrival of a first segment, initializing the linked list of the specified queue;

(2) storing and managing segments according to stages **(a) i)** to **v)**;

vii) upon completion of stage **(b) v)**, concatenating segments as follows:

(1) if no new segments have arrived, copying the snapshot head and tail pointers to the queue linked list head and tail pointers;

(2) if the snapshot linked list were completely emptied, preserving the queue linked list, and holding only the newly arriving segments;

(3) otherwise, concatenating the linked list of the newly arrived segments to the snapshot linked list.

16. A computer program product comprising a computer useable medium having computer readable program code embodied therein for receiving and dispatching data packet segments associated with one or more unicast queues, the computer program product comprising:

computer readable program code for causing the computer to fetch for each arriving segment a structure pointer from a free structure reservoir, and to fetch a data segment pointer from a free data pointer reservoir;

computer readable program code for causing the computer to store the data segment in a memory address pointed to by said data segment pointer;

computer readable program code for causing the computer to store the data segment pointer in the structure pointed to by the structure pointer;

computer readable program code for causing the computer to maintain a packet indicator in the data structure for indicating if the current segment is a first segment or a last segment or an intermediate segment in the packet;

computer readable program code for causing the computer to append the data structure to a structure linked list associated with said queue;

computer readable program code for causing the computer to dispatch stored packet train comprising a specified number of segments, segment by segment, from a specified queue until the packet indicator of the current segment indicates that the current segment is the end of packet, and dispatching all segments

of a successive packet in the queue would result in dispatching more data segments than said specified number of segments said code including:

computer readable program code for causing the computer to create a snapshot of the linked list associated with said specified queue by copying the list head and tail structure pointers to a snapshot head and snapshot tail pointers;

computer readable program code for causing the computer to fetch a data segment pointer from the structure pointed to by the snapshot head pointer,

computer readable program code for causing the computer to dispatch a current data segment pointed to by said data segment pointer;

computer readable program code for causing the computer to update the snapshot head pointer to point to a successive structure in the linked list;

computer readable program code for causing the computer to allow concurrent reception of segments of newly arrived packets to continue, and including:

computer readable program code for causing the computer upon arrival of a first segment to initialize the linked list of the specified queue;

computer readable program code for causing the computer tostore and manage segments;

computer readable program code for causing the computer to concatenate segments and including:

computer readable program code for causing the computer to copy the snapshot head and tail pointers to the queue linked list head and tail pointers if no new segments have arrived;

computer readable program code for causing the computer to preserve the queue linked list, and hold only the newly arriving segments if the snapshot linked list were completely emptied;

computer readable program code for causing the computer to concatenate the linked list of the newly arrived segments to the snapshot linked list if the snapshot linked list were not completely emptied.